CSc 372 — Comparative Programming Languages
Spring 2026 (McCann)

http://www2.cs.arizona.edu/classes/cs372/spring26/

Practice Pre–Final–Exam Homework

(0 points — Just For Practice!)

'Due Date': May 6th, 2026, at the review session

Directions

This practice homework is meant to help you prepare for the final exam by providing questions that cover some (not all!) of the material from the topics we've covered since Exam #2 — specifically, Topics 7, 8, and 9 (Logic Programming Languages, Types, and Conditional Statements).

I recommend that you use these questions in either of two ways, depending on your preparation preferences:

(a) Treat it like Homeworks #2 and #3; that is, write out answers to each question, without the help of AI, to help you understand the material more deeply than merely reviewing the slides would.

(b) Study the material from those topics as you would ahead of an exam, then try to answer these questions as if they were exam questions. Some of these questions are suitable exam questions, while others are probably too long/detailed for that purpose. Even so, using them as potential exam questions will help you judge how well you understand the concepts.

Note that these questions do not ask about everything we covered from these topics (just as Homeworks #2 and #3 did not). Having considered these questions, go back through the material and learn it in similar depth. Then, do the same for the earlier topics of the course — remember, the final is comprehensive!

My suggested answers will be made available after the final exam's review session, to help encourage you to try them before the review session, and to encourage you to attend the review session to help you prepare for the final exam!

Louden/Lambert Chapter 4: Logic Programming

1. Horn Clauses are often expressed as disjunctions rather than implications. Why?
2. Facts are implications without a [?] . A [?] is a body without a head.
3. Show how resolution and unification are used to show that `mortal(socrates)` is true, given the fact `man(socrates)` and the rule $\forall x\,(\text{man}(x) \rightarrow \text{mortal}(x))$.
4. In class I used a family tree example (see the April 1st slides). Consider this new rule, which says that if X is a parent of Y, then X is also Y's ancestor: $\text{parent}(X, Y) \rightarrow \text{ancestor}(X, Y)$. Given the fact `parent(ben,carl)`, use unification and resolution to show that `ancestor(ben,carl)` is true.
5. We covered Prolog's `is/2` predicate in class. What does the "2" tell us?
6. How are the fact `parent(ben,carl)` and the logical implication $\forall x(\text{parent}(x, y) \rightarrow \text{ancestor}(x, y))$ expressed in Prolog syntax?
7. We can express the idea that a value is a power of 3 recursively:

   Initial Condition: 1 is a power of 3 ($3^0 = 1$)

   Recurrence Relation: If n is a power of 3, then $n * 3$ is also a power of 3.

   Create a Prolog database that tests non–negative integers to see whether or not they are powers of 3.

(Continued . . .)

8. How is the list `[[1],[2,3]]` represented as a binary tree in Prolog?

9. If we have the Prolog fact `list([[1],[2,3]]).` , what is the result of the query `list([H|T]).` ?

10. The 'cut' predicate in Prolog.

    (a) What is the purpose of 'cut' in Prolog?

    (b) Prolog purists are not big fans of 'cut'. Why not?

11. Consider this pair of Prolog rules:

```
biggerer(F, S, F) :- F >= S.

biggerer(F, S, S) :- F < S.
```

    The invocation `biggerer(5,4,X).` correctly finds that X = 5, but wants to keep searching for another answer that cannot possibly exist. Show how to modify this pair of rules using 'cut' to keep the pointless additional searching from occurring (while still producing correct answers!).

12. In class we covered the function and implementation of Prolog's `member/2` predicate. What does `member/2` do for us, and what does `member(v,[v,i,v,i,d]).` return, assuming that we press `;` whenever we are able?

Louden/Lambert Chapter 8: Data Types

13. How is code readability improved by strongly–typed languages?

14. How does *type checking* differ from *type inferencing*?

15. In class I defined strongly–typed languages, and said that languages that fail to meet that high standard are often called "weakly–typed." Then I defined dynamically–typed languages. Do a little research before you try to answer this question (and then answer it!): How are weakly–typed and dynamically–typed languages distinguished from each other?

16. A simple type supports single values. Enumerated types are defined by listing (enumerating) all possible values that the type represents. This may make it seem that enumerated types are not simple, but they are. Explain why enumerated types are considered to be simple.

17. Why is the `union` type in the C language considered to be based on the set operator 'union' instead of the set operator 'Cartesian Product' as C's `struct` type is?

18. Consider the assignment statement $a = b*10$. Explain how the Hindley–Milner type inferencing algorithm can infer appropriate types for the variables a and b.

Louden/Lambert Chapter 9: Control I — Expressions and Statements

19. Why does the Böhm–Jacopini theorem include iteration but not recursion?

20. Explain how the Guarded IF structure is a generalization of modern IF–THEN, IF–THEN–ELSE, and CASE (a.k.a. SWITCH) statements.

Answers will be provided a few days before the final exam.
In the meantime, try answering them yourself!